Kansas State's Slick Willie Robot Software

David A. Gustafson

■ Robotics Team 1 from Kansas State University was the team that perfectly completed the Office Navigation event in the shortest time at the fifth Annual AAAI Mobile Robot Competition and Exhibition, held as part of the Thirteenth National Conference on Artificial Intelligence. The team, consisting of Michael Novak and Darrel Fossett, developed its code in an undergraduate software-engineering course. Its C++ code used multiple threads to provide separate autonomous agents to solve the meeting scheduling task, control the sonar sensors, and control the actual robot motion. The team's robot software was nicknamed SLICK WILLIE for the way it gracefully moved through doorways and around obstacles. The resulting code was robust and performed excellently.

Kansas State University Robotics Team 1 tied for second in the Office Navigation event at the fifth Annual Mobile Robot Competition and Exhibition, held as part of the Thirteenth National Conference on Artificial Intelligence (AAAI-96). The team, consisting of Michael Novak and Darrel Fossett, accomplished the complete Office Navigation event perfectly. In both the second and the final rounds, the software achieved the maximum points for successfully completing the event without hitting obstacles, hitting walls, incorrectly estimating the time for the meeting, or failing to enter rooms. The time for completion of the task was less than one-third the time of the only other team to perfectly complete the task.

Novak and Fossett's software planned a route from the director's office to the conference rooms, directed the robot to each of the two conference rooms, correctly determined which conference room was not occupied, planned a route to the other offices and back to the director's office, guided the robot to each of the other offices, estimated the time for the meeting and announced the time of the meeting, and then directed the robot back to the director's office exactly one minute before the announced time for the meeting.

Novak and Fossett (figure 1) were undergraduate computer science students in the Computing and Information Science Department at Kansas State University (KSU) in Manhattan, Kansas. During the 1995 to 1996 academic year, they took a two-semester, software-engineering course that I taught. In this required course, the principles of software engineering are taught using a team-project approach. Their project was to develop software on the Nomad 200 robot for tasks such as maze following, office delivery, and office navigation. The course emphasized the development of robust software and the adequate testing of the software in a variety of situations.

After completion of the course in May 1996, Novak and Fossett specialized their team's code for the AAAI-96 robotics competition. By the end of June, they had robust software for the competition's Office Navigation event.

The Robot

The robot, named WILLIE after the KSU wildcat mascot, is a Nomad 200 robot from Nomadic Technology, Inc., in Sunnyvale, California. The robot is approximately two feet in diameter and three feet tall. It is equipped with 2 sonar rings of 16 sonars each and with 2 charge-coupled device (CCD) cameras. The robot has a 486 processor on board with a hard drive and 16 megabytes of memory. The turret and the

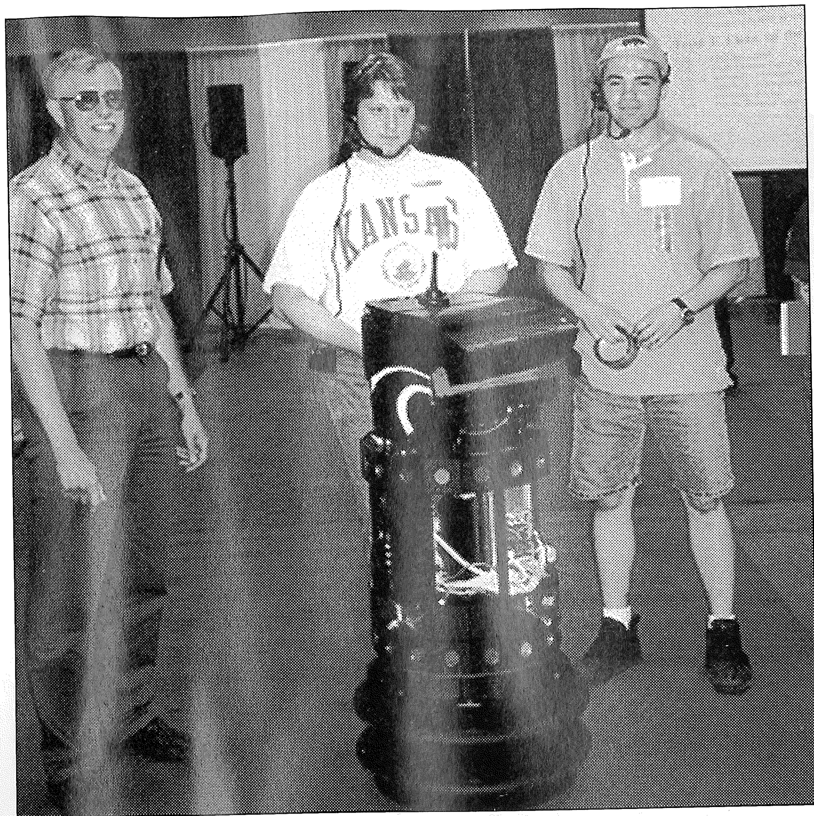


Figure 1. David Gustafson, Michael Novak, and Darrel Fossett.

base can rotate independently. The robot can translate forward and backward on its three wheels, which rotate together. The robot has a zero turning radius and a maximum speed of 20 inches/second. The Nomad 200 robot has a simulation system that runs on UNIX workstations. This simulation system allows extensive execution of the software in a variety of situations.

The programming was done in C++ under the LINUX operating system that was installed on the 486 PC on board the Nomad 200. Although the team used a workstation and wireless ethernet to initiate the execution of the robot software, the Nomad 200 was run as an autonomous robot under the control of the software on board the robot.

The Design

The object-oriented approach provided robust software to control the robot. The main principle of the software development was to partition the tasks and responsibilities into top, middle, and low levels. For example, the top level needed to know about what the Office Navigation event was and how to solve it but did not need to know about obstacle avoidance. The behaviors at the bottom level needed to worry about low-level responsibilities, such as avoiding obstacles and not hitting walls but did not need to know about the overall strategy for solving the task. Figure 2 shows a simplified version of the object model. Most of the attributes and some of the methods are not included. The classes marked with an asterisk run as separate threads.

The top-level–object *high_level_control* was responsible for the sequencing of the path-planning, high-level motion, observation, estimation, and direction tasks and for high-level error recovery. The high_level_control object handles the direction of motion down hallways and into and out of rooms, but lower-level objects control the local movement.

Below the top level are objects to handle the mapping functions, the detection functions, and the robot motion. The midlevel-object *map* is responsible for the mapping and path-planning functions.

The midlevel object *detect* is responsible for using the camera to detect motion in the conference rooms. The low-level–object *camera* controls the actual camera. In each room, the camera was aimed in three directions, overlapping the whole area of the room, and in each of the directions, a sequence of camera images was compared by the detect object to identify any movement or change in the image.

The last midlevel object, *motion*, is responsible for general hallway motion, the exiting of rooms, and the entering of doorways.

The move function in the motion object does two important tasks (figure 3). The first task is to avoid moving objects. It calls the function *blocked* to check on the immediate area in the direction of motion. If there is an obstacle to forward motion, it waits and retries. If there is not an obstacle or if the obstacle has moved, it continues to monitor the areas directly in front of the robot. It plans the local movement necessary to avoid obstacles. It also uses a low-level object, *vector,* to identify the immediate free area in front of the robot, allowing the robot to smooth out the motions necessary for moving through doorways and around obstacles. The team's software controller was nicknamed SLICK WILLIE because its motion through doorways was a graceful sliding motion instead of the boxy movements common in robots centering on, and moving through, doorways.

The low-level object *vector* is responsible for maintaining the local view of the surround-

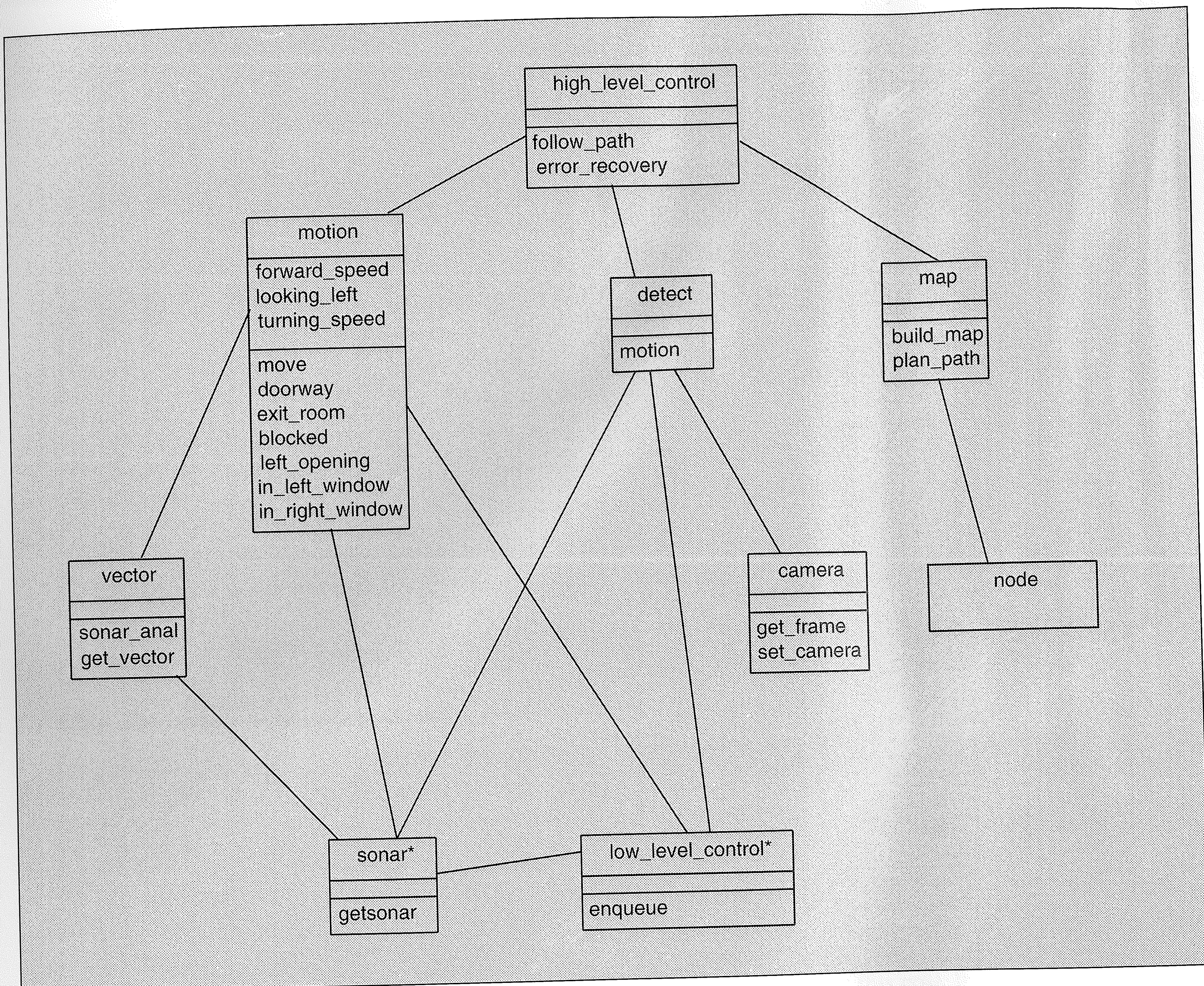


Figure 2. Simplified Object Model of Control Software.

ings. It uses a trigonometric analysis of the sonar readings to determine the locations of walls, and so on. It stores a description of the local space. It can also be used to produce a virtual wall on one side of the robot to allow simple wall following when appropriate. Using this local space description, the *get_vector* function returns the best local direction to move.

Also in the motion object is the blocked function. It is the heart of collision avoidance. Blocked determines if the area in front of the robot is free. It checks the sonar readings and determines if there is a possibility of forward or sideways movement. Blocked is called by all the functions that do midlevel movement control.

The team used threads to provide autonomous agents for the low-level control and sonar functions. The low-level–object *sonar* runs continuously and provides assurance of the detection and avoidance of obstacles, stationary or moving. The sonar object also filters the raw sonar data to prevent erroneous transitory sonar readings from confusing the higher-level objects. Sonar is also responsible for returning the current condition to other objects.

The actual movement of the robot is under the control of the low-level–object *low_level_control*. The high-level and midlevel objects do not directly send commands to the robot's motors. The object low_level_control maintains a queue of motion requests from

```
while (1) {

  // see if intended path of travel is blocked
  if (blocked())
    wait a few seconds - say "get out the way"
    if (blocked())
      return (failure);

  if (left_opening()) and (in_left_window()) and (looking_left)
    return (success);

  if (right_opening()) and (in_right_window()) and !(looking_left)
    return (success);

  // forward speed - based on a linear function dependent upon how close to obstacles
  forward_speed = get_forward_speed();

  // get the direction to go - this will always be a vector toward the best "freespace" area in
  // the intended movement direction
  turning_angle = vector->get_vector();

  // the turning speed is a linear function based upon how far we have to turn
  turning_speed = get_turning_speed( turning_angle );

  // enqueue the move
  low_level_control->enqueue( vm, forward_speed, turning_speed, 0 );
}
```

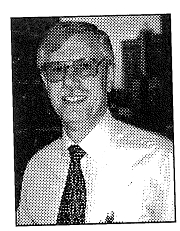
Figure 3. Algorithm for Move Function in Class Motion.

the higher-level objects. It decides on which request to process and how to implement the request.

The result of this design is a robust system that has partitioned the tasks among appropriate objects and has delegated the responsibilities to objects at an appropriate level. This approach allows each object to have a simple approach to solving the problems for which it is responsible. The resulting code is clear and understandable, and the software system is robust.

The Students

Michael Novak is now a graduate student at KSU, working on a master's degree in computer science, and Darrel Fossett completed his bachelor's degree in December 1996 and is currently interviewing for a software development position.



David A. Gustafson is a professor in the Computing and Information Sciences Department at Kansas State University. His teaching interests include software engineering, robotics, and expert systems. He is the author of more than 30 papers in software measures and software engineering. He received his Ph.D. in computer science from the University of Wisconsin at Madison.